A New Approach to Computer Performance Prediction

John Gustafson, Ames Laboratory, US DOE

Abstract.

What is the ultimate goal of computer performance prediction? In brief, it is to find a quick, inexpensive, and reasonably accurate answer to this question: "How well will a given computer run a given application, and why?" At Ames Laboratory, we have found a promising approach based on a hardware characterization called *HINT* and a corresponding software characterization called the *application signature*, that decouples the analysis problem in an intuitive manner. It provides a visual way to judge, for example, whether a problem is suited to a particular parallel computer. It can also assist hardware engineers in the design of computers for particular applications.

1. Introduction

Try to imagine studying physics without a measure of length other than "Object A looks larger than Object B," or studying chemistry without a measure of temperature better than "The reaction vessel became very hot." Without absolute metrics, sciences remain in a state of infancy. That is the state that computer science has been in.

Conventional measures of computer performance do not have the rigor of, say, meters per second. [1]. As a result, we have poor means of comparing algorithms and poor means of comparing hardware systems. A near-universal fallacy is that this can be corrected by comparing only differences in execution time, leaving everything else the same. But performance measurement almost universally means changing precision, reliability, cost, memory size, algorithm, and architecture simultaneously! The single-number ratios one obtains by comparing execution times shed very little light on the question: "How well will this computer run this problem?"

2. HINT

In 1994 and 1995, Quinn Snell and I developed a graphical measure of performance called HINT that fixes neither the execution time nor the problem size, and is based strictly on what a computer accomplishes instead of its nominal activity. "HINT" is rapidly gaining acceptance as a rigorous means of assessing computer performance, where performance incorporates everything that contributes to the quality of an answer.

For a complete explanation of the principles of HINT, refer to [3] or to the Web page `http://www.scl.ameslab.gov/HINT.` For purposes of this discussion, we need only a feel for the meaning of HINT graphs. Figure 1 shows a typical graph for a RISC workstation.

*This work is supported by the Applied Mathematical Sciences Program of the Ames Laboratory-U.S. Department of Energy under Contract W-7405-ENG-82.

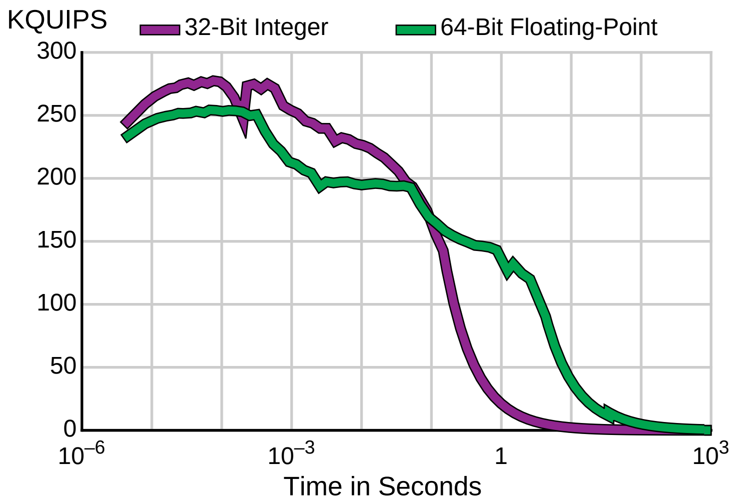

Figure 1. Generic HINT Graph

The vertical axis shows QUIPS, or "QUality Improvement Per Second." In an ideal situation, the quality obtained after *N* iterations is *N*. Actual quality degrades because of precision loss. The rate at which iterations is performed tends to drop because the amount of memory demanded increases, leading to the use of slower regimes in the memory hierarchy. One can use integer or floating-point arithmetic, of *any precision desired*, selecting the tradeoff between fast-and-approximate or slow-and-precise to give optimal results. The native data type that gives best performance is an indication of the personality of the computer, since scientific computers are typically optimized for floating-point math but business and personal computers are typically better at dealing with characters and integers.

The horizontal scale is *time*, not problem size as one so often sees in other benchmarks. This is crucial. Consider what physical velocity under constant acceleration looks like on a time graph compared with what it looks like on a distance graph:

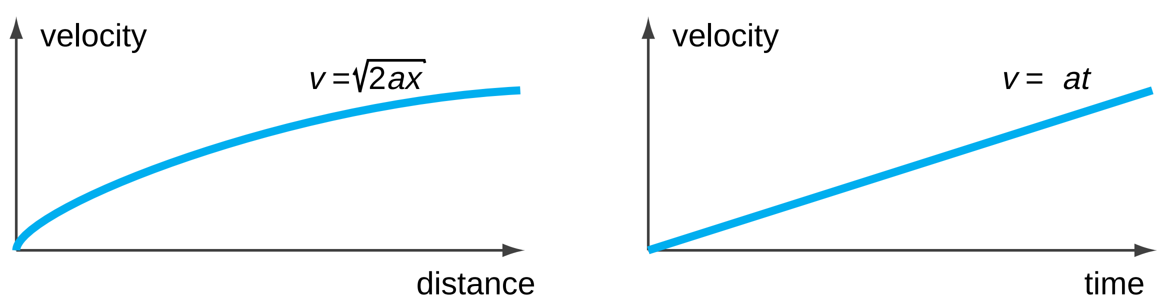

Figure 2. Wrong and Right Ways to Graph Physical Speed

Scaling requires comparison in temporal terms [2]. As computers vary in capability, their work is increased so that humans perceive roughly constant delays for tasks. From pocket calculators to supercomputers, human patience for answers spans the same range even though the work done by the machine may vary by a factor of a billion! By Occam's razor, the use of time as the independent variable feels right to us. We use a logarithmic scale because of the importance of every time range from nanoseconds to seconds in assessing performance.

Let’s return to Figure 1. HINT graphs show characteristic “regimes” of performance. On the left one sees the speed of primary cache (and of course, registers). At some point there is a “cliff” where performance drops suddenly to a lower level. If there is a secondary cache, there will be another “cliff” as the problem size grows to require main memory. If the system has virtual memory on mass storage, that shows up on the rightmost part of the HINT graph. **HINT lays out the speed of the computer hardware as a function of the problem size attempted**. It is the only benchmark to make this clear. Most benchmarks, such as SPEC, select a problem size and imply that the resulting speed applies to any problem size. No computers have flat performance as a function of problem size. This is perhaps the main way that people are misled by benchmarks.

When a benchmark ranks two computers one way but someone gets the opposite ranking using a real application, the usual excuse is, “No one benchmark can capture the operation mix of every application.” While this is true, it probably isn’t the reason for the error. A more likely explanation is that **the benchmark sampled one part of the HINT curve, and the application sampled a different part**. See Figure 2. This reasoning leads to the half of performance prediction complementary to the HINT hardware graph: the application signature.

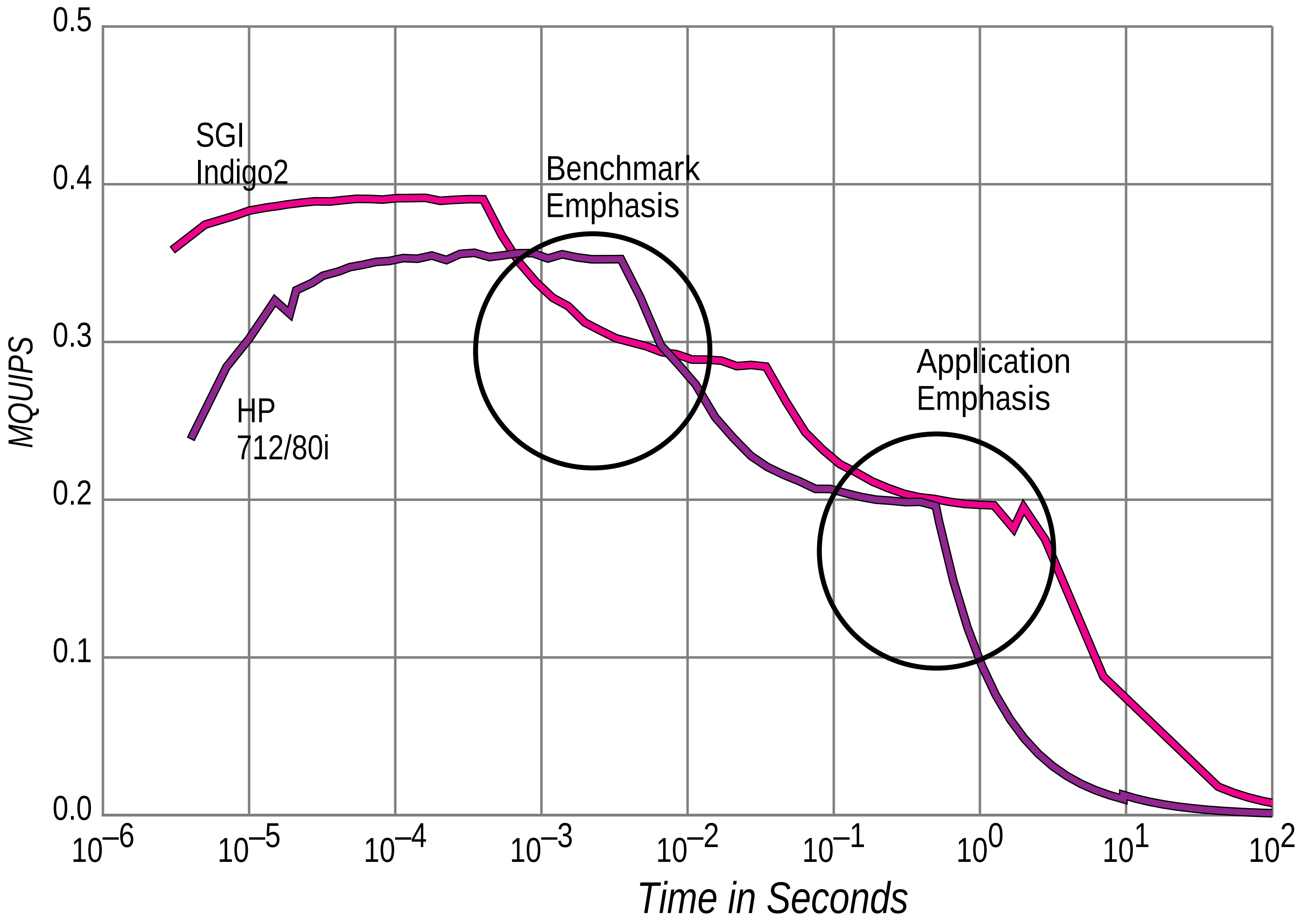

Figure 2. One Reason Benchmarks Mispredict Performance

The Application Signature

This paper marks the first presentation of the "application signature" concept. Suppose one looks at the memory access pattern of an application using the same time scale as the HINT graph, where the vertical axis is now a density function. One can imagine attaching a logic analyzer to the memory bus of a computer, and taking the power spectrum of the pattern of addressing. Tiny loops will cause spikes on the left, representing tasks that easily fit in registers or cache. Larger loops will have a period in the millisecond to second range. Is this strictly a function of the computer on which the application is run? No; we claim that this signature has much more to do with the application and the user environment than with the particular computer.

Consider the example of word processing. The application signature probably looks much like the following:

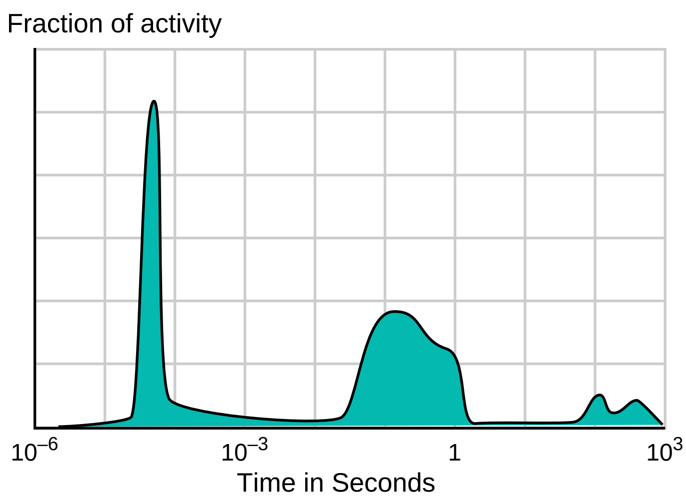

Figure 4. Estimated Signature Graph for Word Processing

When the computer is in a polling loop waiting for a keyboard or mouse event, its activity is a spike on the left part of the signature. Since a human types at about 3 to 10 keystrokes per second, another component of activity will always show up in that 100 ms to 300 ms range. Finally, for the occasional save to disk, global search, word count, print, or scroll through the document, there will be events in the several second range.

Will these spikes shift left or right or change in respective height as one goes from one computer to another, or from WordPerfect® to Microsoft Word®? Yes, but probably not very much. They occur where they do and as much as they do because of the demands and time scales of human thought and human fingers performing the editing function.

Consider a very different type of application: A computational fluid dynamics (CFD) simulation. Regardless of the specific method used, these simulations invariably use as much main memory as is available, use that memory to represent the state of the fluid, and compute time steps by updating that memory according to physical rules. We can reason out what the application signature graph should look like:

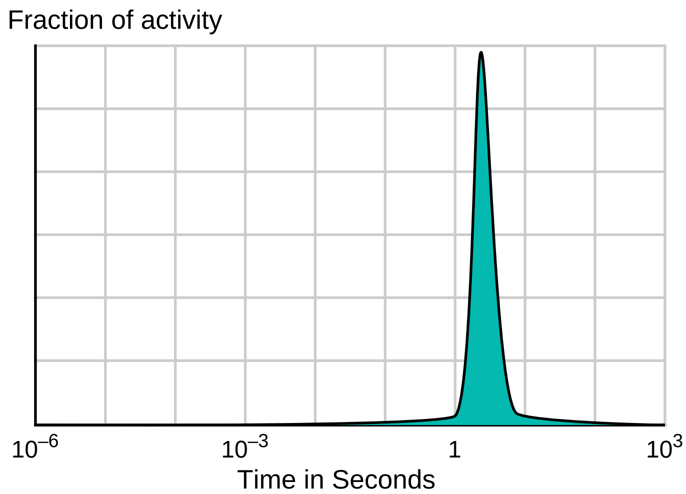

Figure 5. Estimated CFD Application Signature

It has been said that scientific problems do not cache well. This is because the programs tend to traverse the entire memory, over and over, for the reasons just stated. There is little need to pound repeatedly on just a few memory locations as in the word processing application.

The Prize: Real Performance Prediction

The punch line should now be obvious: To predict performance, simply multiply the HINT graph by the application signature!

This does more than answer the question, “How long will it take to execute this task?” It says *how well suited a given computer system is to a task, from the standpoint of actual use*.

Analogies

Consider the following three types of vehicle: A four-door sedan; a racing car; an all-terrain vehicle. Each has a time scale in which it excels in performance. The all-terrain vehicle might be the best at stop-start driving with chaotic maneuvers in the range of one second. The sedan might do best at city driving in the range of 15 second intervals of acceleration and deceleration. The racing car is presumably optimized for long straight runs at maximum speed, but might not brake or turn tight corners particularly well. The equivalent of the HINT graph would be speed versus time. The equivalent of the application signature would be the stop-start pattern of a given travel path (through traffic, obstacles, or racetrack conditions).

Perhaps an even more direct analogy is that of stereo speaker response curves. Here is a curve from a recent copy of *Consumer Reports*:

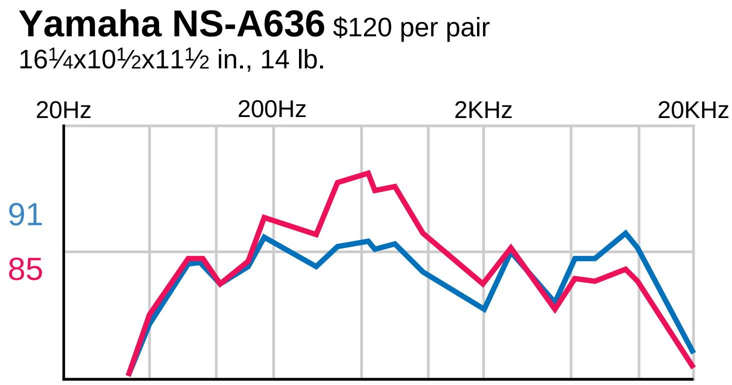

Figure 6. Acoustic Speaker Performance

The logarithmic horizontal scale is now reciprocal time (frequency), but conveys the same physical idea. These curves are very similar to HINT graphs. Now suppose one has a need to reproduce the human voice, bird calls, or an earthquake. Each emphasizes a different range, and its power spectrum is very much like the application signature graph. The analogy breaks down in that flat response is desirable for sound systems, whereas a HINT graph should be as high as possible, whether flat or not.

Application to Parallel Computing

Much has been said and written about scalability, speedup, the limits of parallel processing, and how difficult it is to know when to use a parallel computer versus a traditional uniprocessor. The methods proposed here make it easy to assess the merits of parallel processing.

The following HINT graph is for an nCUBE 2S running 1, 2, 4,…, 256 processors. The case of 1 processor is a serial version of HINT with no message-passing. Note how different it is in character from the others in that it extends much farther to the left.

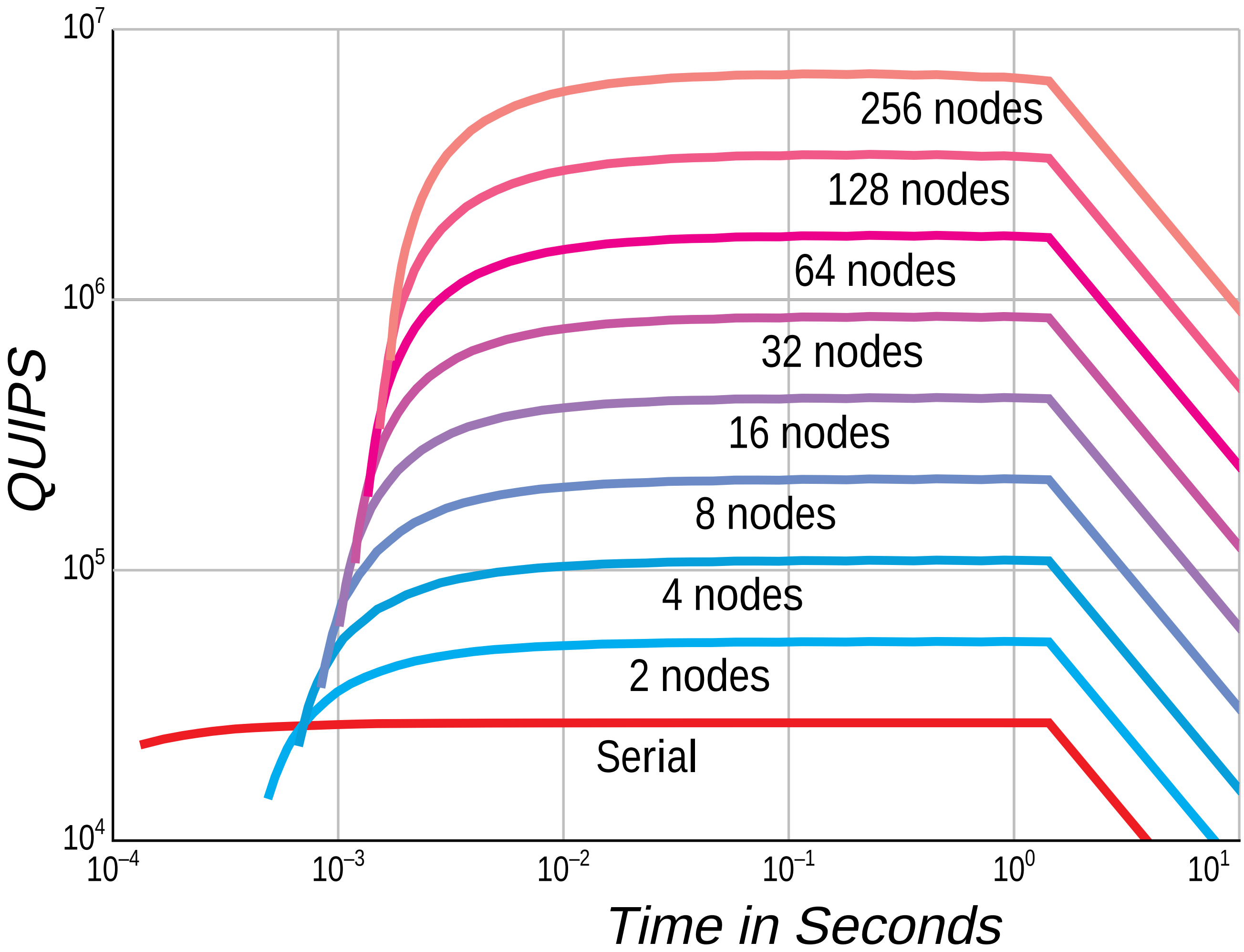

Figure 7. HINT graphs for the nCUBE 2S Parallel Computer

While one might be impressed by the height of the curves for large numbers of processors, we've already seen that an application signature might have the bulk of its distribution on the leftmost part of the curve where the large ensembles do poorly. The word-processing application makes little sense to run on even two processors! Only if there was an unusual situation where there was many global search-and-replace operations to do would parallel processing provide a benefit. On the other hand, the CFD signature shows an excellent fit to parallel computing capability. The main spike in CFD occurs just before one runs out of main memory, which is also where the performance hits its maximum.

If one were to scale a problem down in size, one can find even a CFD problem so small that it would run better on a single processor than in parallel. But it would run in a ridiculously small amount of time and give answers of little scientific value. A fallacy in the "Amdahl's law" type of reasoning is to think that humans run a fixed size problem on all different computer sizes, which would lead to great inefficiency. By plotting things versus time, and recognizing that human time scales largely set the location of the spikes in the application signature, one better understands why parallel computers have proved so much more useful than argued by Amdahl. More justification for the use of fixed-time concepts can be found in [2].

Designing Computers to Fit

This method of analysis provides a good estimator for hardware designers trying to match the needs of particular applications. Most experienced computer engineers have an intuitive feel for what is and is not worthwhile to add to a system, but these graphs can help out the less experienced and perhaps give more precise information to an engineer asked to build for an application about which he or she knows very little.

It is said that Seymour Cray never put virtual memory on his vector supercomputers because, he said, "On machines this fast, there's no use pretending you've got something that you haven't." What he knew intuitively was that most of his customers had application signatures like that of the CFD application. Hence he never used cache or automatic paging from disk in his architectures.

Conclusions

Part of this paper is based on a mature, actual measurement called HINT that succinctly describes the performance of a piece of hardware in a way that seems largely application-independent. The other part is currently a thought experiment and a work in progress… the "signature graph" of an application, something one can estimate now and which we soon expect to measure using logic probes and digital spectrum analyzers on the memory bus. Both are predicated on the observation that memory references dominate both system cost and algorithm content. We have thus shifted the emphasis in computer performance analysis from floating-point operation speed (which has become a notoriously poor indicator of anything), to memory speed, size, and latency at all levels from registers to mass storage. We now seek the experimental evidence to show that HINT in combination with application signatures can be an accurate predictor of computer performance. If we are successful, the performance prediction and analysis problem will have been successfully factored into a hardware description and a software description that need only be multiplied together to produce a practical guide to application/architecture fit. ■

References

[1] J. Dongarra and W. Gentzsch, Editors, *Computer Benchmarks*, North-Holland, 1993.

[2] J. Gustafson, "The Consequences of Fixed-Time Performance Measurement," *Proceedings of the 25th Hawaii International Conference on System Sciences*, 1990.

[3] J. Gustafson and Quinn Snell, "HINT: A New Way to Measure Computer Performance," *Proceedings of the Twenty-eight Annual Hawaii International Conference on System Sciences*, Vol. II, pp. 392–401, 1995.